Towards Multi-Layer Autonomic Isolation of Cloud Computing and Networking Resources

Aurélien Wailly (aurelien.wailly@orange-ftgroup.com)*
Marc Lacoste (marc.lacoste@orange-ftgroup.com)*
Hervé Debar (herve.debar@telecom-sudparis.eu)†

Abstract: This paper describes a flexible approach to manage autonomically cloud resource isolation between different layers of an IaaS infrastructure, reconciling computing and network views. The corresponding framework overcomes fragmentation of security components and automates their administration by orchestrating different autonomic loops, vertically (between layers) and horizontally (between views).

Keywords: cloud computing, autonomic security, resource isolation

1 Introduction

Despite its many foreseen benefits, the main barrier to adoption of cloud computing remains security. Vulnerabilities introduced by virtualization of computing resources, and unclear effectiveness of traditional security architectures in fully virtualized networks raise many security challenges [7]. The most critical issue remains resource sharing in a multi-tenant environment, which creates new attack vectors. The question is thus how to guarantee *strong resource isolation*, both on the computing and networking side. System and network complexity make manual security maintenance by human administrators impossible. Computing and networking isolation over virtualized environments should thus be achieved by automated means.

Unfortunately, current solutions fail to achieve that goal: hugely fragmented, they tackle the problem only from the computing or the networking side, and moreover at a given layer – thus without end-to-end guarantees. This is particularly true for IaaS infrastructures, where different heterogeneous security components may be involved at the hardware-, hypervisor-, or VM-level, making the overall security infrastructure difficult to manage. A new integrated and more flexible approach is therefore needed.

This paper describes a unified autonomic management framework for IaaS resource isolation, at different layers, and from both computing and networking perspectives. We propose a nested architecture to orchestrate multiple autonomic security loops, both over views and layers, resulting in very flexible self-managed cloud resource isolation. A first design for the corresponding framework is also specified, and implemented for a simple IaaS infrastructure. The possibilities of such an architecture are illustrated on a sample scenario including virus detection, isolation, and cleaning phases.

* Orange Labs
† Télécom SudParis

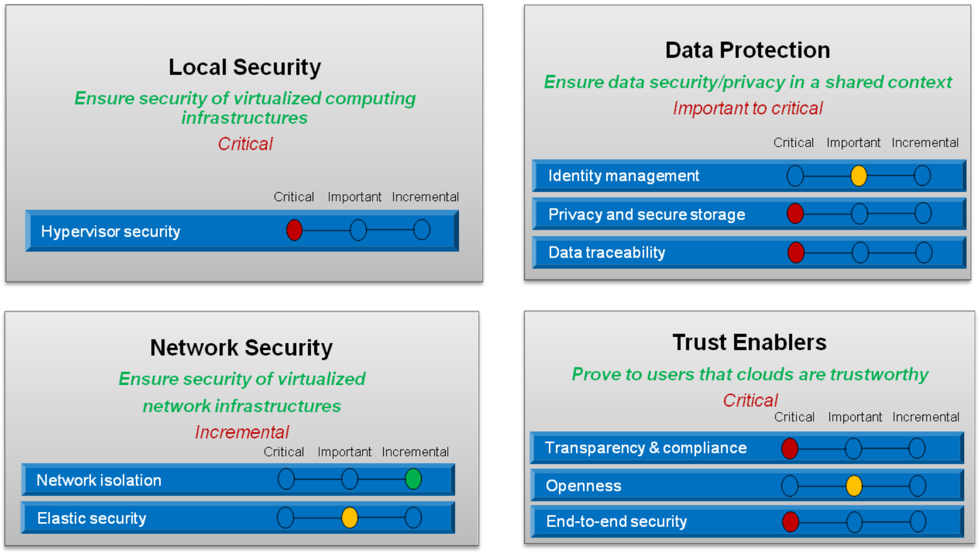
Local Security
Ensure security of virtualized computing infrastructures
Critical
Critical Important Incremental
Hypervisor security
Data Protection
Ensure data security/privacy in a shared context
Important to critical
Critical Important Incremental
Identity management
Privacy and secure storage
Data traceability
Network Security
Ensure security of virtualized network infrastructures
Incremental
Critical Important Incremental
Network isolation
Elastic security
Trust Enablers
Prove to users that clouds are trustworthy
Critical
Critical Important Incremental
Transparency & compliance
Openness
End-to-end security

Fig. 1: Major Security Roadblocks of Cloud Computing.

The rest of this paper is organized as follows. After a survey of current cloud security challenges (Section 2), we focus on resource isolation, reviewing related work (Section 3). We then present our approach for autonomic isolation of cloud resources (Section 4), along with a first cloud isolation framework (Section 5). We also describe its simple implementation on a typical IaaS infrastructure (Section 6). The paper concludes by describing ongoing work (Section 7).

2 Cloud Security Challenges

Nine major roadblocks of cloud computing security may be identified to capture the security barriers to adoption of cloud infrastructures. Those roadblocks may be classified into the 4 main areas shown in Figure 1: *local security* (protection of computing resources), *network security* (protection of communications), *data protection* (protection of storage), and other *trust enablers*. These roadblocks are thorny challenges which will require a multi-disciplinary approach to be solved, taking into account organizational, technical, and business perspectives. Their criticality may be qualified using the following scale:

- *Critical:* lifting the roadblock is essential for adoption, resulting in a breakthrough if successful.
- *Important:* lifting the roadblock will be a major step forward.
- *Incremental:* lifting the roadblock is possible by natural enhancement of already existing technologies.

In what follows, we discuss for each area the corresponding security roadblocks.

2.1 Local Security

This area deals with protection of the servers which compose a data center. The main issue is how to guarantee security when computing resources are virtualized, i.e., as VMs running above a hypervisor on each host. The key roadblock is thus naturally the *security of the hypervisor*.

Hypervisor Security. Virtualization introduces many security vulnerabilities. Clouds are by essence multi-tenant environments: the crux is thus strict isolation between VMs, which may fail if the hypervisor is compromised. In theory, the most widespread hypervisors have a relatively low surface of attack. In practice, new variety of attacks [17, 18] such as installing rootkits inside the hypervisor (*hyperjacking*) or using covert channels call for higher degrees of assurance [19]. The main weaknesses are misconfigurations, malicious device drivers, and backdoors between the VM and the hardware, issues for which there are today no real answers. Hypervisor security is only part of the problem since VMs may also bring their own set of vulnerabilities: these may be mitigated using hardened VM images, or strict VM security life-cycle management. In all cases, "security by default" configurations should be applied, with clear delineation of responsibilities between customer and cloud vendor.

2.2 Network Security

This area deals with the protection of communication channels to access or inside a data center. The key issue is how to guarantee security when networking resources are virtualized, i.e., firewalls, IDSes, routers run as virtual appliances. The main roadblocks here are *network isolation* and *elastic security.*

Network Isolation. In a data center where some of the communication links may be fully virtualized, are traditional network security architectures still effective? Where should security controls be placed? The risks are broadly comparable to those of traditional networks (confidentiality/integrity of network connections to/in clouds, AAA, availability). The corresponding counter-measures (encryption, digital signatures, NAC, VPNs, NIDS/NIPS,...) are thus still applicable. The main change is that network isolation is no longer physical but logical: network zones where traffic could be segregated physically (e.g., to separate production from supervision hosts) are replaced with logical security domains, where traffic between VMs is filtered by "virtual" firewalls. As a result, isolation is less precise, and the security guarantees weaker. Overall, the technical security components are available today to lift this roadblock. However, the main difficulty is to map them to cloud architectures.

Elastic Security. Flexible allocation and rapid provisioning of security resources able to respond to dynamic evolutions in the cloud is still a challenge due to the high rate of change in virtual servers. First solutions are emerging for flexible and dynamic management of VPNs, with the notion of virtual private clouds. However, fully automated security supervision of cloud infrastructures is still lacking due to the complexity and short response times needed to manage vulnerabilities, detect intrusions, and activate defenses. Research initiated by IBM on autonomic, self-protecting security architectures should enable to build infrastructures where security is self-managed, security parameters autonomously being negotiated with the environment to match the ambient estimated risks and provide an optimal level of protection [11].

2.3 Data Protection

This area deals with the security and privacy of data (at rest and in transit). The main issue is how to guarantee such security in a shared, multi-tenant environment. Key roadblocks are both technical such as *identity management*, but also non-technical, such as *privacy/secure storage* or *data traceability*.

Identity Management. The number and diversity of principals using cloud services internally and externally, and the volume of resources accessed call for end-to-end solutions for managing identities. Unfortunately, cloud infrastructures are still lacking consistent identity information architectures: multiple administrators, credential repositories, and processes which are neither automated nor orchestrated may induce new vulnerabilities. Barriers are thus scalability, heterogeneity and interoperability. An unsolved problem is how to achieve federation of identities across data centers, organizations, or cloud providers to avoid duplications of identities, or privileges between information systems.

While the basic mechanisms already exist with standards like SAML to exchange identity claims, how to integrate them in the cloud context is still unknown. If authentication challenges might be overcome in the near future, uniform management of authorizations is still in its infancy despite standards like XACML. Nonetheless, identity management is one of the main opportunities for Security as a Service in cloud infrastructures, to outsource authentication and authorization components from IT systems.

Privacy and Secure Storage. Today's privacy concerns will be magnified in cloud environments due to technology sharing in a multi-tenant context. The main challenge is thus strong data isolation throughout the life-cycle of personal information. Difficult problems include data access enforcement on a need-to-know basis, secure data storage and information flow control, and data retention and destruction. While standard cryptography is still applicable to protect data at rest and in motion, today's privacy best practices are not enough to fully address this roadblock. Possible technical answers include self-destructing data [9], "sticky" (i.e., directly attached to the data) privacy policies, and more generally promoting "privacy by design". In any case, responsible data stewardship in the cloud requires both in-depth understanding of possibilities of privacy-enhancing technologies (PETs), and of legal implications in contractual agreements.

Data Traceability. Adding further to the loss of control of user over their data, locating the data itself is a major concern in a shared and virtualized infrastructure: at a given time, a cloud provider might not know exactly where (i.e., in which country) data is stored, processed, or accessed from. Without special care, data could move freely around between organizations, or even between international borders. This raises legal and political issues, since several jurisdictions (EU Data Protection Directive, US Safe Harbor Program) specifically require the provider to have such knowledge. Data hosted abroad might also be exposed to foreign governments (USA Patriot Act). Data traceability is also needed to prove to users that data comes from a trusted source. Overall, this domain is still a widely unchartered area.

2.4 Trust Enablers

This last area is perhaps the most important one, since the main issue is how to prove to customers that their cloud infrastructure is trustworthy. The main roadblocks are *transparency and compliance*, *openness*, and how to guarantee *end-to-end security*.

Transparency and Compliance. Unlike traditional environments, in the cloud, the nebula of stakeholders, logical instead of physical isolation, and transfer of resources outside the control of organizations, all make customers uneasy as where trust boundaries have moved to. A maximum level of transparency is thus required from the vendor to dispel this confusion. Customers need tangible evidence of the security hygiene of a provider infrastructure, to verify security claims of contractual agreements, or compare with other providers practices. Elements of assurance may also be required by authorities to check compliance with established standards and regulations. Unfortunately, providers remain so far opaque on these aspects.

Auditability of infrastructures should thus be enhanced, to convince third parties that the necessary detective and preventive security controls are in place. Setting up a certification process may help the provider move forward. Unfortunately, there is no real agreement today on the right assurance framework (SAS 70, ISO 27001,...). Documents published by ENISA [8] or the Cloud Security Alliance [6] may facilitate a risk analysis. Yet this analysis remains difficult due to increased complexity and openness compared with traditional computing. Trusted computing technologies [2] may also foster trust by providing users cryptographic evidence of infrastructure integrity, but a lot of work remains to be done in this area. In any case, responsibilities between providers and customers should be established using clear-cut SLAs.

Openness. This issue is necessary to overcome vendor lock-in, viewed by the Cloud Security Alliance (CSA) as a top threat. Proprietary, closed, and non standard-compliant cloud technologies will make it very complex for the customer to change cloud provider, verify the vendor security promises, or react in case of incident, forensic support not necessarily being available. Interoperability with other cloud infrastructures will also be hard, each vendor having its own APIs. Deployment of applications distributed across several infrastructures will thus be hampered, limiting scalability. Recommendations are to stick to best practices [6, 8] and standards, and push standardization efforts on open APIs. Open source cloud architectures will also bring additional benefits in terms of flexibility (modular architectures) and security (careful code scrutiny by the security community).

End-To-End Security. To foster trust, data isolation should be guaranteed both at rest and in transit in all levels of the cloud infrastructure (processing, network, storage). Unfortunately, the few security building blocks available are highly heterogeneous and fragmented. How to orchestrate them seamlessly into an end-to-end security infrastructure for cloud environments is still undefined. This issue which crosscuts all the previous technological barriers could be overcome by standardizing reference security architectures for cloud environments which describe the organization of the different security components, in order to provide an overall view of cloud security.

Although most of those roadblocks are critical, in this paper, we tackle the problem of IaaS resource isolation, both from the computing and networking perspectives, thus addressing the *hypervisor security*, *network isolation*, and *elastic security* roadblocks in the first two areas. We also address the *end-to-end security* roadblock, by defining building blocks for a reference cloud security framework.

Layer	Networking View	Computing View
Application (VM)	Application-level firewall: WAF... Virtual firewall: VShield Zones/App... SSL/TLS VPN	Antivirus: VShield EndPoint VM introspection [4]
Hypervisor	Virtual switch [5, 15, 13] System-level firewall: iptables, ebtables L2/L3 VPN IP overlay network [12]	Security API: [14] Security modules: [2, 20]
Physical	Dedicated network equipment: firewall (VShield Edge), physical switch, router VLAN L1 VPN Link overlay network [21]	MMU/IOMMU Hardware-assisted virtualization: Intel-VT, AMD-V VNICs

Tab. 1: Some Solutions for Cloud Resource Isolation.

3 Related Work

In the cloud, pooled networking and computing resources may be seen from two different but complementary views. The *networking view* abstracts the network resources, i.e., the successive protocol layers that encapsulate the data to be transmitted in the cloud. Orthogonally, the *computing view* captures the computational and storage resources of each machine at different abstraction levels (software and hardware), e.g., processor, memory, devices. Ensuring end-to-end isolation requires a fine-grained control of information manipulated in each layer crossed along the data path.

To simplify, we consider three main layers in an IaaS infrastructure: *physical*, *OS* (hypervisor), and *application and/or middleware* (VM-level), broadly corresponding to OSI network layers 1 and 2, 2 to 4, and 3 to 7.

Virtualization opens totally new attack vectors, as shown recently by many loopholes exploited in each layer. Compromising the hypervisor by a side-channel attack may leak information between VMs sharing resources [17]. Layer spoofing is also possible as in the BluePill rootkit [18]. Another possibility is to fully bypass intermediate layer controls. For instance, the guest OS might directly access virtualized devices. Similarly, the applicative layer could attempt to spoof ARP requests to break physical isolation (e.g., MAC filtering). Next, we provide a brief overview of some existing isolation mechanisms in each layer, summarized in Table 1.

Physical Layer. At this level, network isolation relies on dedicated network equipments like switches, routers, and firewalls. Besides evident physical separation, IEEE 802.1Q-compliant Virtual Local Area Networks (VLAN) enable to segregate virtual networks on the same physical infrastructure. It is also possible to create network overlays at link layer [21] or above [12]. Firewalls can filter desired packets in a fine-grained manner using Access Control Lists (ACLs). They can be configured from a console via a serial port, or by accessing a tiny Web server directly integrated into the equipment. The computing view is far more complex with hardware-virtualized resources, a technology that allows a single physical equipment to offer seamlessly several instances to the OS, such as Network Interface Controllers (VNICs) [22], Input/Output Memory Management Units (IOMMU) that map device-visible virtual addresses to physical ones, and special instruction sets to deliver virtualization in processors (Intel-VT, AMD-V).

Hypervisor Layer. One level up, the hypervisor provides control with software-virtualized network interfaces and firewalls. The networking view is richer with kernel modules that extend physical switches into virtual ones. The two main competing solutions are Open vSwitch [15] and Cisco Nexus 1000v [5]. Both are very close, but Open vSwitch implements OpenFlow [13] to offer precise control of forward tables going well beyond ACL-based administration. Further control on communications is possible thanks to system-level firewalling solutions, e.g., iptables filtering rules for IP packets, ebtables rules for the link layer, or solutions like VMware VShield Zones.

In the computing view, resource isolation can be performed through hardware-assisted instructions or emulation. Type 1 hypervisors seem most suitable for fine-grained control over resources. Some generic hypervisors are actively maintained, with both open source and commercial solutions. VMware ESX is a mostly closed platform, control remaining limited to a fixed set of APIs, and thus difficult to extend. The Kernel-based Virtual Machine (KVM) and Xen use Intel-VT or AMD-V for instruction processing, IOMMU to separate resources, and deliver a device driver to manage virtualization hardware. KVM implements VMs as Linux processes, and therefore benefits from a wide variety of standard tools for process isolation and management. Security modules [2, 20] for isolation are available for Xen based on SELinux – thus directly supported by KVM with the advantage of built-in isolation of Linux kernel backend drivers. The libvirt library is also increasingly used to administer a wide range of hypervisors and modules. It provides a higher-level hypervisor-independent interface to control the virtualization environment. Security modules for this library are available, e.g., sVirt [14] providing MAC security schemes to isolate guest OSes and specify access permissions.

VM Layer. The VM essentially relies on the hypervisor capabilities for computing and networking isolation. However, a growing number of virtual appliances are already available to filter network data (virtual firewalls such as VShield Zone/App from VMware), to monitor the VM security status (VM introspection [4]), or to isolate a group of compromised VMs by defining a quarantine zone (antivirus suites). These solutions run into conventional user/group administration problems such permission management to determine the domains in which users, applications, and devices can be added. Some of those solutions provide little or no explicit interface to manage remotely the other VMs. Moreover, in the guest OS, security control usually remains limited to userland, while kernel modules provide richer and stronger isolation.

Overall, those solutions suffer from three main limitations:

1. Available mechanisms are highly heterogeneous, lacking of an overall architectural vision regarding their orchestration into an integrated security infrastructure.

2. The different security configurations of previously underlined mechanisms (e.g., VM-level firewall rules and physical-level ones) induce scalability and maintainability challenges: even if a cloud environment may be tuned to meet specific needs, nested dependencies between views and layers can become appalling and virtually impossible to solve, excluding the "by hand" approach to security management.

3. The extremely dynamic character of the cloud (e.g., with VM live migration between physical machines), and the short response times required to activate system defenses efficiently, make the problem even more complex.

A flexible, dynamic, and automated security management of cloud isolation mechanisms is thus clearly lacking today.

4 Autonomic Isolation of Cloud Resources

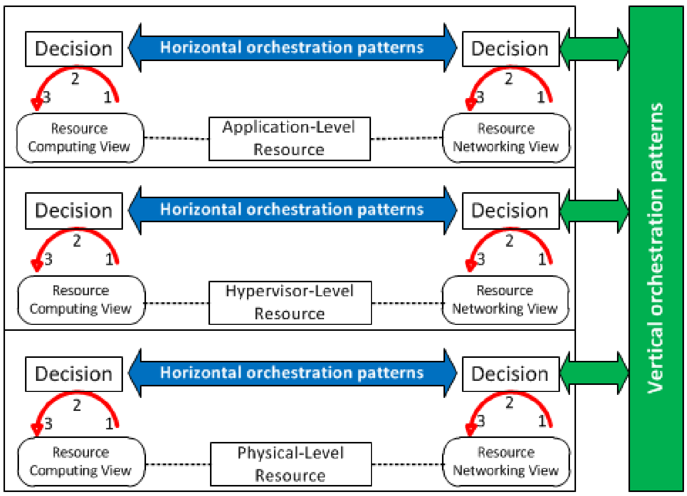
Decision
Horizontal orchestration patterns
Decision
Resource Computing View
Application-Level Resource
Resource Networking View
Hypervisor-Level Resource
Physical-Level Resource
1
2
3
Vertical orchestration patterns

Fig. 2: Multiple Autonomic Loops Orchestration for Resource Isolation.

The previous limitations may be overcome by adopting an *autonomic approach to security* of IaaS infrastructures [3], to get over fragmentation of security components providing resource isolation, automate security administration, and react rapidly to detected threats. Thus, security becomes self-managed through control loop patterns in the isolation mechanisms throughout the system. This means introducing *detection*, *decision*, and *reaction* security components that collaborate to select the adequate isolation policies matching the ambient estimated risk, and achieve an optimal level of protection of cloud resources.

We propose to apply that approach for cloud resource isolation, both in terms of *views* and *layers*. To synchronize the different corresponding autonomic loops, we distinguish two types of *loop orchestration*, *horizontal* and *vertical*, as shown in Figure 2:

- The autonomic paradigm may be applied to different *views* of cloud resource isolation, both computing and networking. Indeed, earlier research on autonomics is usually separated between *autonomic computing*, providing self-management capabilities to overcome growing system complexity, but taking distribution of resources for granted, and *autonomic networking*, applying similar principles to communication environments to master rising network management complexity. An end-to-end cloud security framework should thus be able to reconcile both views. *Horizontal orchestration patterns* are thus introduced to coordinate autonomic computing and networking loops between the two views.

- Autonomic behavior can also be considered at the different IaaS *layers*. Depending on the targeted mechanism, a great number of autonomic loops may thus be introduced, which may be increased due to cross-layering issues. In order to keep the self-management model consistent, those loops have to be synchronized to maintain a stable state. This is the role of the *vertical orchestration patterns* which coordinate the loops between the different layers.

Several patterns may be used for orchestrating autonomic loops – in particular for communication between different loops [1]. A *centralized* model is perhaps the simplest way to organize autonomic components: a single entity dictates the global behaviour to other autonomic managers. This pattern is extensible to a *hierarchy* of autonomic managers. Other patterns such as *peer-to-peer* are also possible, but out of the scope of this paper, where only simple orchestration patterns will be considered.

Having views interacting with layers opens a totally new range of possibilities in terms of flexibilility for cloud isolation. The whole security management model is engineered by orchestrators that obey to generic patterns, specified by the administrator. Such a design makes possible the implementation of high-level strategies for cloud resource isolation, enabling easy administration and high dynamicity.

For instance, consider a scenario where a physical network equipment realizes isolation between applicative VM resources, also supervised by a virtual security appliance. When an attack is detected, in the physical layer, a first loop may decide to isolate a specific host in a VLAN. At the application level, the security appliance may also decide to update its database containing signatures of known viruses. However, if connection to the network is lost, the second loop will not work properly, a timeout being triggered. For the architecture to remain consistent, our approach enables to introduce rules for the physical layer loop to notify the application layer loop to stop network interactions.

5 Isolation Framework Design

We now give a brief overview of our proposition for an autonomic management framework for cloud resource isolation orchestrating feedback loops over views and layers based on the previous principles.

5.1 Framework Overview

As shown in Figure 2, our autonomic isolation framework makes the following assumptions on the IaaS infrastructure: (1) each cloud resource offers well-defined interfaces/hooks to capture its state (*detection*), and to perform actions on it (*reaction*), both from the system and the network perspectives; (2) in each layer, individual decision-making components (i.e., autonomic managers) provide the necessary interfaces to realize the collective layer management behavior (*horizontal orchestration*); and (3) each layer offers the needed management interfaces to perform synchronization between layers (*vertical orchestration*).

Two types of orchestrators guarantee overall consistency of self-management of cloud resource isolation:

- *Vertical Orchestrators (VO)* synchronize overall cloud resource isolation management behaviors between layers, and implements the vertical orchestration patterns. For instance, in case of inconsistency between layers, a VO may decide to enforce specific rules on well-chosen layers, according to administrator-defined policies.

[1] Communication interfaces between different loops may be specified using an Interface Description Language (IDL). Thus, each loop building block may be developped in any language provided it matches the interfaces defined in the IDL, enabling low-level detection/reaction components to communicate beyond the scope of a single loop with high-level decision-making components.

- *Horizontal Orchestrators (HO)* synchronize in each layer cloud resource isolation behaviors between computing and networking views of resources. Each view sends collected informations to the HO (1). The HO summarizes its knowledge and provides it to VO (2). When a local layer isolation policy has to be modified (or new policies chosen at the overall level), views are updated by the HO through hooks previously specified (3).

Thus, orchestrators implement a hierarchical and layered self-management model, distributed over layers and views, and defining a modular system that can be easily extended to fit particular cloud network architectures.

5.2 A Sample Use Case

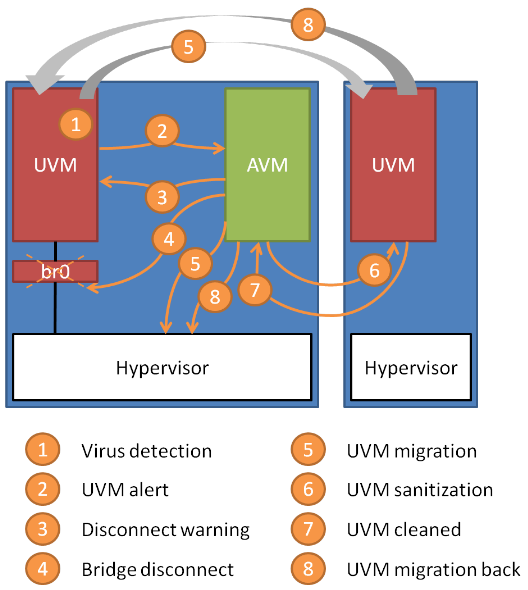

Fig. 3: A Simple Use Case.

A typical example that underlines the framework interest is shown in Figure 3:

1. A User VM (UVM) detects the presence of a virus that can compromise nearby VMs.
2. An alert message is sent to the VM-layer HO.
3. The HO sends back to the UVM an "about to disconnect" event.
4. The HO chooses to isolate the compromised VM in both networking and computing views, by cutting the network link `br0` in the underlying hypervisor.
5. The HO also decides to migrate VM ressources on a physically separated computer specially designed for this purpose.

6. Then, once isolation is completely achieved, the VM-layer HO sends an order to trigger the cleaning of the user VM.
7. The sanitization status of the UVM is sent back to the HO upon cleaning completion.
8. This allows the hypervisor-layer HO to reassign original VM resources, and migrate the VM back outside the quarantine zone.

This simple example shows how to perform active defense on such an architecture.

6 Implementation

6.1 IaaS Framework Instantiation

Figure 4 represents a simple implementation of the isolation framework on a typical IaaS infrastructure. Dedicated network equipments provide the *physical architecture*. Network traffic is segregated by a firewall by ACL rule-matching, by a switch through VLAN tables, and by routing tables. All of these security policies are modifiable by the physical autonomic loop. For our implementation, two VLANs are plugged between the firewall and the physical machine.

The *hypervisor* (KVM) contains Linux-specific policies, such as internal routing tables, or memory associations between physical and virtual devices. In the figure, `peth0` represents the physical Ethernet interface, `eth0` and `eth1` are physically virtualized interfaces, while `br0` and `br1` are the bridge abstractions needed by the hypervisor to switch on/off an interface. Bridge `br0` memory is simply associated with `eth0`, as `br1` with `eth1`. Each bridge will be the endpoint of an emulated interface for VMs. The libvirt API handles the remote access to establish the hypervisor- layer autonomic loop.

At the *VM layer*, we consider two types of VM: the *administrative VM (AVM)* and the *user VM (UVM)*. A UVM contains at least two components: a firewall, to isolate network flows, and an antivirus that take cares of data execution prevention, program isolation and kernel signals. In the antivirus kernel component, probes monitor the loaded images in the guest OS. UVMs communicate with the hypervisor through the `conn1` connection endpoint, connected to `br0` – a second VM would be connected through `br1` and so on. The UVM virtual memory is mapped to the physical machine memory. They run on the virtual CPU (vCPU) abstraction provided by the hypervisor.

The AVM collects probes from every layer and organizes framework decisions with orchestrators (vertical and horizontal). The AVM behaves as a security management interface, collecting threat information, and deploying counter-measures. In each layer, the autonomic managers (HO_X) negotiate with both a centralized VO, and with the layer management APIs.

6.2 Framework Implementation

At the hypervisor layer, the libvirt API uses a library named netcf to enforce new network rules via XML. Although the frontend is clearly defined, the backend is OS-dependent: we thus have to fully translate netcf's XML configuration files and to implement commands for interface creation, modification and deletion. Actually, bridge modifications are fully implemented in order to have a first concrete example (use case described in Section 5.2).

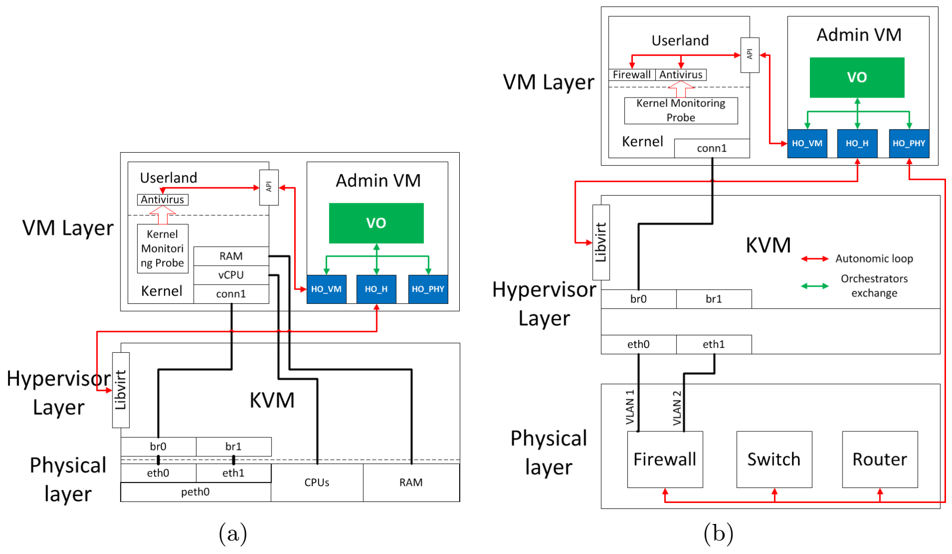

Fig. 4: IaaS Framework Instantiation: (a) Computing View; (b) Networking View.

In the VM layer, we are using ClamAV as a flexible antivirus with Python support for remote control as the source code is available. Real-time protection is missing, so we implemented a kernel module to scan files when they are loaded in memory thanks to `PsSetLoadImageNotifyRoutine` and control their execution by defining a `PsSetCreateProcessNotifyRoutine` that can collect ClamAV results with I/O request packet (IRP) and act accordingly. This implentation underlines what can be achieved in the VM layer: specific functions such as filtering socket creation to ban a range of compromised VMs can also be hooked.

Communications in heterogeneous environments require clearly-specified interfaces, e.g., using an IDL. Due to its good results (see the benchmarks of [1]), we chose the Google IDL implementation named protobuf[10] to implement communications between the HOs and the different layer components, and with the VO. To implement the VM-layer components, the C language was naturally used, as low-level programming is needed for the UVM. However, the HOs and VO were chosen to be implemented in Python, as those components only need to take decisions on a high-level.

This particular implementation is actually under deployment as security infrastructure for the French government-funded SelfXL project, aiming at self-management of large scale systems such as cloud computing infrastructures. It allows the realization of dynamic quarantine zones to isolate and clean potentially compromised VMs.

6.3 Use Case Implementation

The implementation of the use case defined in Section5.2 required two main features: (1) to easily control bridges created by KVM for VMs; and (2) to migrate VMs through physical equipments with libvirt. Bridge control can be achieved in many ways, but we will focus on the following methods:

- Each newly created VM is directly connected to a `vnet` sub-interface, all of them being bridged together to a single bridge. This is the classic way to perform such a task, but it resides on the capacity of KVM to handle the network. Unfortunately, during our tests we were unable to recover connectivity after deleting a `vnet` interface from the bridge.
- For each VM created, a virtual interface is created at the hypervisor layer. A bridge is also linked to this sub-interface and will be one end of the VM connection. Sub-interfaces can be Ethernet abstractions provided by IP aliasing, or KVM `vnetX` interfaces. This approach, although more complex during the creation process, does not suffer from any major problems. Creation and deletion are totally independent, and are based on classic Linux networking operations.

To properly migrate VMs, all hypervisor interface names are synchronized. This task is handled by orchestrators that manage an association table between VM and network names.

Communication between AVM and UVMs while the network is down can be solved in many ways, around one common idea: establish a shared zone.

- Just as VMware and VirtualBox install their add-ons, communication can be achieved by emulating the insertion of a CD-ROM. If mounted as read and write, it provides an easy buffer to transfer data back to the hypervisor.
- Instead of cutting the wire directly, the action can be to isolate the VM in a specific VLAN. This VLAN contains a network storage (or equivalent) that only handles and delivers simple messages.
- Virtual Machine Introspection [16] (VMI) techniques also provide VM monitoring directly through the hypervisor.

With such techniques, the antivirus can find patches to a virus that was not clearly identified before the network isolation operation.

7 Ongoing Work

This paper described a flexible approach to manage autonomically cloud resource isolation between different layers, reconciling computing and network views. The corresponding framework overcomes fragmentation of security components and automates their administration by orchestrating different autonomic loops, vertically and horizontally.

We are actually working on a semantic representation to express rules and manage objects in an easy way – for instance to describe orchestration pattern strategies, or interactions between loop components. A well-choosen set of verbs in that semantics will simplify and make more precise administration rules.

Going beyond the presented architecture requires several modules in different views, currently under implementation. HTTP Web servers included into physical equipments need specific APIs and a wrapper in the AVM. VMI is also a rising technology: we are currently evaluating features of available products, but the VM manipulation directly from KVM remains an opaque matter. We will go deeper inside KVM source code in order to estimate what can be done.

Acknowledgments

This work has been funded by the ANR SelfXL project.

References

[1] thrift-protobuf-compare: Comparing Various Aspects of Serialization Libraries on the JVM Platform. `http://code.google.com/p/thrift-protobuf-compare/`.

[2] S. Berger, R. Careces, D. Pendarakis, R. Sailer, and E. Valdez. TVDc: Managing Security in the Trusted Virtual Datacenter. *ACM SIGOPS Operating Systems Review*, 42(1), 2008.

[3] D. Chess, C. Palmer, and S. White. Security in an Autonomic Computing Environment. *IBM Systems Journal*, 42(1):107–118, 2003.

[4] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra, and D. Zamboni. Cloud Security is not (just) Virtualization Security. In *ACM Workshop on Cloud Computing Security (CCSW)*, 2009.

[5] Cisco. Nexus 1000v. `www.cisco.com/web/go/nexus1000v`.

[6] Cloud Security Alliance. Security Guidance for Critical Areas of Focus in Cloud Computing. `http://www.cloudsecurityalliance.org/csaguide.pdf`.

[7] Cloud Security Alliance. Top Threats To Cloud Computing. `http://www.cloudsecurityalliance.org/topthreats.html`.

[8] ENISA. Cloud Computing: Benefits, Risks and Recommendations for Information Security, 2010.

[9] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *USENIX Security Symposium*, 2009.

[10] Google. protobuf Serializer. `http://code.google.com/p/protobuf/`.

[11] R. He, M. Lacoste, and J. Leneutre. A Policy Management Framework for Self-Protection of Pervasive Systems. In *International Conference on Autonomic and Autonomous Systems (ICAS)*, 2010.

[12] X. Jiang and D. Xu. VIOLIN: Virtual Internetworking on OverLay INfrastructure. In *International Symposium on Parallel and Distributed Processing and Applications*, 2004.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Computer Communication Review*, 38:69–74, March 2008.

[14] J. Morris. sVirt: Hardening Linux Virtualization with Mandatory Access Control. In *Linux.conf.au Conference*, 2009.

[15] Open vSwitch. `openvswitch.org`.

[16] J. Pfoh, C. Schneider, and C. Eckert. A Formal Model for Virtual Machine Introspection. In *Workshop on Virtual Machine Security (VMSec)*, 2009.

[17] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My Cloud! Exploring Information Leakage in Third-Party Compute Clouds. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[18] J. Rutkowska and A. Tereshkin. Bluepilling the Xen Hypervisor. In *BlackHat Technical Security Conference (BLACKHAT)*, 2008.

[19] J. Rutkowska and R. Wojtczuk. The Qubes OS Architecture. Technical report, Invisible Things Lab, 2010.

[20] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. van Doorn. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *Annual Computer Security Applications Conference (ACSAC)*, 2005.

[21] A. Sundararaj and P. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *USENIX Virtual Machine Research and Technology Symposium (VM)*, 2004.

[22] S. Tripathi, N. Droux, T. Srinivasan, and K. Belgaied. Crossbow: From Hardware Virtualized NICs to Virtualized Networks. In *ACM Workshop on Virtualized Infrastructure Systems and Architectures*, 2009.